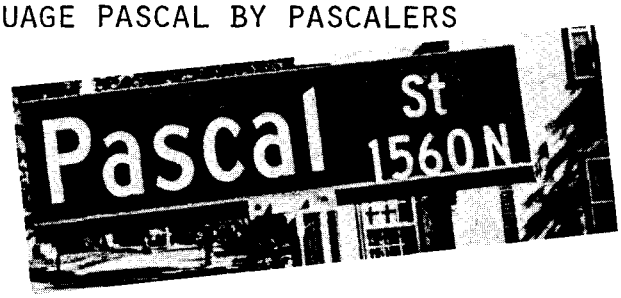
PASCAL USER'S GROUP

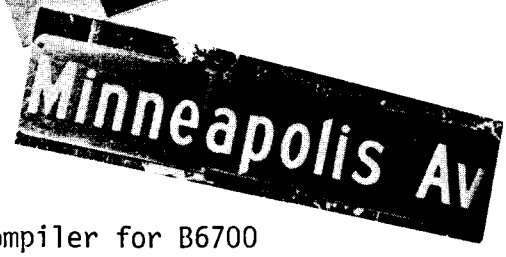
Pascal News

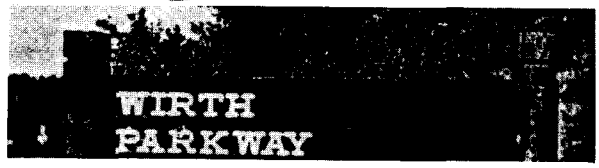
October, 1979 Number 16

SPECIAL ISSUE ON THE PASCAL VALIDATION SUITE

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS







EX LIBRIS: David T. Craig
736 Edgewater
[#_____] Wichita, Kansas 67230 (USA)

POLICY: PASCAL NEWS (79/09/01)

* Pascal News is the official but informal publication of the User's Group.

  Pascal News contains all we (the editors) know about Pascal; we use it as the vehicle to answer all inquiries because our physical energy and resources for answering individual requests are finite. As PUG grows, we unfortunately succumb to the reality of (1) having to insist that people who need to know "about Pascal" join PUG and read Pascal News - that is why we spend time to produce it! and (2) refusing to return phone calls or answer letters full of questions - we will pass the questions on to the readership of Pascal News. Please understand what the collective effect of individual inquiries has at the "concentrators" (our phones and mailboxes). We are trying honestly to say: "we cannot promise more than we can do."

* An attempt is made to produce Pascal News 3 or 4 times during an academic year from July 1 to June 30; usually September, November, February, and May.

* ALL THE NEWS THAT FITS, WE PRINT. Please send material (brevity is a virtue) for Pascal News single-spaced and camera-ready (use dark ribbon and 18.5 cm lines!).

* Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

* Pascal News is divided into flexible sections:

  POLICY - tries to explain the way we do things (ALL-PURPOSE COUPON, etc.).

  EDITOR'S CONTRIBUTION - passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.

  HERE AND THERE WITH PASCAL - presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal in the news, history, membership rosters, etc.

  APPLICATIONS - presents and documents source programs written in Pascal for various algorithms, and software tools for a Pascal environment; news of significant applications programs. Also critiques regarding program/algorithm certification, performance, standards conformance, style, output convenience, and general design.

  ARTICLES - contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.)

  OPEN FORUM FOR MEMBERS - contains short, informal correspondence among members which is of interest to the readership of Pascal News.

  IMPLEMENTATION NOTES - reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Portable Pascals, Pascal Variants, Feature-Implementation Notes, and Machine-Dependent Implementations.

* Volunteer editors are (addresses in the respective sections of Pascal News):

  Andy Mickel - Outgoing editor; Rick Shaw - Incoming editor
  John Eisenberg - Here and There editor
  Rich Stevens - Books and Articles editor
  Bob Dietrich and Gregg Marshall - Implementation Notes editors
  Jim Miner and Tony Addyman - Standards editors
  Andy Mickel and Rich Cichelli - Applications editors
  Jenny Sinclair and Rick Marcus - Tasks editors

- - - - - - ALL-PURPOSE COUPON - - - - - - (01-Dec-79)

Pascal User's Group, c/o Rick Shaw
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342 USA

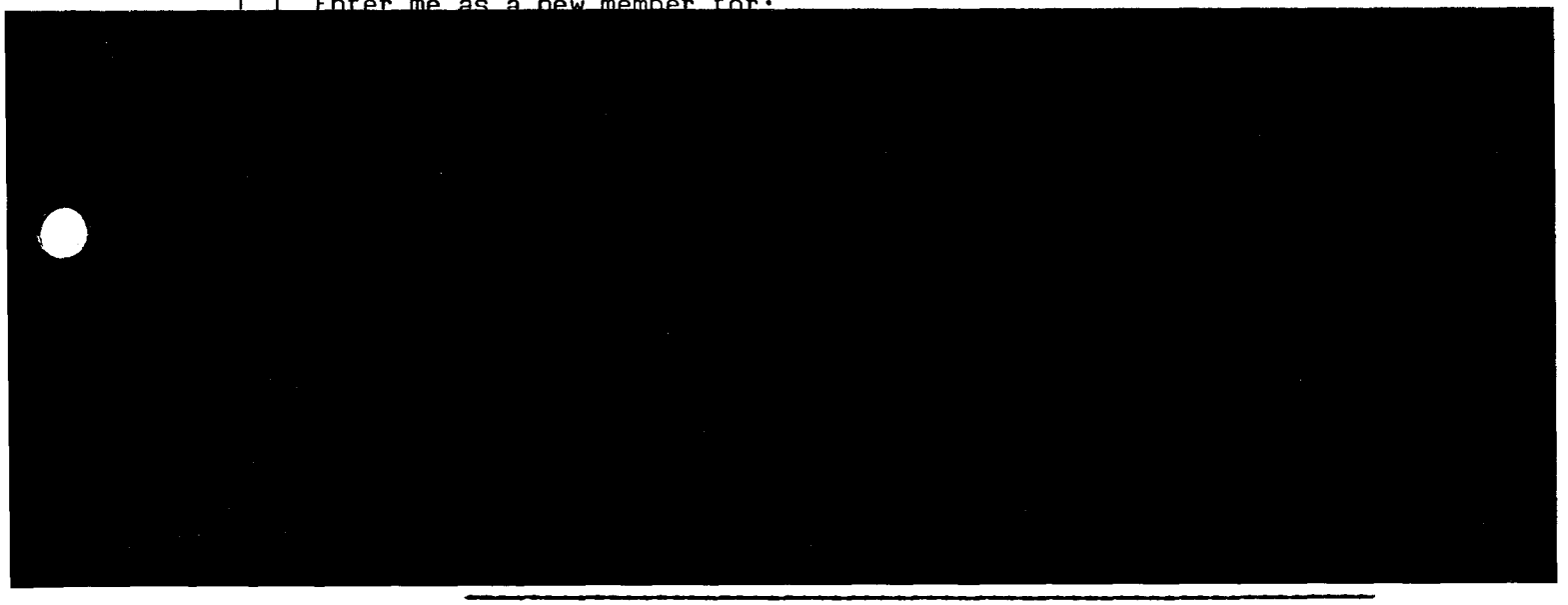
NOTE

- Membership is for an academic year (ending June 30th).
- Membership fee and All Purpose Coupon is sent to your Regional Representative.
- SEE THE POLICY SECTION ON THE REVERSE SIDE FOR PRICES AND ALTERNATE ADDRESS if you are located in the European or Australasian Regions.
- Membership and Renewal are the same price.
- The U. S. Postal Service does not forward Pascal News.

- -

[ ] 1 year ending June 30, 1980

[ ] Enter me as a new member for:



_____

ENCLOSED PLEASE FIND:	$	
	A$	
	£	____.____

NAME _____

ADDRESS _____

_____

_____

PHONE _____

COMPUTER _____

DATE _____

JOINING PASCAL USER'S GROUP?

- Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan.
- Please enclose the proper prepayment (check payable to "Pascal User's Group"); we will not bill you.
- Please do not send us purchase orders; we cannot endure the paper work!
- When you join PUG any time within an academic year: July 1 to June 30, you will receive all issues of Pascal News for that year.
- We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

- American Region (North and South America): Send $6.00 per year to the address on the reverse side. International telephone: 1-404-252-2600.
- European Region (Europe, North Africa, Western and Central Asia): Join through PUG (UK). Send £4.00 per year to: Pascal Users' Group, c/o Computer Studies Group, Mathematics Department, The University, Southampton SO9 5NH, United Kingdom. International telephone: 44-703-559122 x700.
- Australasian Region (Australia, East Asia - incl. Japan): Join through PUG(AUS). Send $A8.00 per year to: Pascal Users' Group, c/o Arthur Sale, Department of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001, Australia. International telephone: 61-02-23 0561.

PUG(USA) produces Pascal News and keeps all mailing addresses on a common list. Regional representatives collect memberships from their regions as a service, and they reprint and distribute Pascal News using a proof copy and mailing labels sent from PUG(USA). Persons in the Australasian and European Regions must join through their regional representatives. People in other places can join through PUG(USA).

RENEWING?

- Please renew early (before August) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News. Renewing for more than one year saves us time.

ORDERING BACK ISSUES OR EXTRA ISSUES?

- Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a academic year (July 1 to June 30) means that we eliminate many requests for backissues ahead of time, and we don't have to reprint important information in every issue--especially about Pascal implementations!
- Issues 1 .. 8 (January, 1974 - May 1977) are out of print.
  (A few copies of issue 8 remain at PUG(UK) available for £2 each.)
- Issues 9 .. 12 (September, 1977 - June, 1978) are available from PUG(USA) all for $10.00 and from PUG(AUS) all for $A10.
- Issues 13 .. 16 are available from PUG(UK) all for £6; from PUG(AUS) all for $A10; and from PUG(USA) all for $10.00.
- Extra single copies of new issues (current academic year) are: $3.00 each - PUG(USA); £2 each - PUG(UK); and $A3 each - PUG(AUS).

SENDING MATERIAL FOR PUBLICATION?

- Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. "All The News That's Fit, We Print." Please send material single-spaced and in camera-ready (use a dark ribbon and lines 18.5 cm wide) form.
- Remember: All letters to us will be printed unless they contain a request to the contrary.

UNIVERSITY OF MINNESOTA
TWIN CITIES
University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

Editor's Contribution

Special Issue

This special issue on the Pascal Validation Suite was prepared primarily by Jenny Mizielinski (now Jenny Sinclair) and Arthur Sale of the Department of Information Science at the University of Tasmania. We owe special thanks to Jenny because she does most of the work for PUG Australasia, and she should have been listed as a "tasks editor" long before now--in fact as far back as 1977!

The Validation Suite represents a valuable weapon in Pascal's arsenal because it provides a common measuring instrument for standards conformance. As has been said before, Pascal now joins a small and elite group of programming languages which has such a collection of test programs.

Note that the Validation Suite is copyright, but that it can be obtained very easily and inexpensively from three (3!) worldwide distributors! ANY user of ANY Pascal compiler ANYWHERE should not pass up this opportunity!

Rick Shaw

Next issue (PN#17) will be Rick Shaw's first issue as editor and is now scheduled for December. (This issue is my last one as editor.) Please don't be alarmed that Rick works for DEC. He will keep PUG and DEC strictly separated. (Besides he is just as funny and crazy a person as I am!) As I said in my editorial in PN #15, Rick is a capable administrator (whereas I am not good at delegating responsibility), and he has the luck of being in a nice work environment at DEC's Atlanta Regional Office with ready access to clerical facilities, etc. We were able to recruit section editors (listed on the inside front cover)to whom Rick can now distribute the work of Pascal News. Decentralization is manditory if Rick is to survive my fate. Good luck, Rick!

About a year and a half ago it would have been hard for me to say goodbye to Pascal News. Now it is really easy! I'm really weary and "burned-out" having worked hard all year--even after having said in issue #13 that I was tired of doing the job. Also I'm comforted that as of about a year ago,it became undeniably obvious that together all of us Pascalers permanently established Pascal as a major programming language--not just "another" language. All progress since then has been and will be pleasant dividends.

Rick's volunteering to be editor for 2 years comes just in the nick of time: a reluctant editor such as myself doesn't contribute to the quality of Pascal News. Rick will provide fresh ideas whereas I'm running out of ideas.

I feel relieved to be rid of the day-to-day responsibility for Pascal News and PUG. (However I intend to help Rick every way I can.) I do admit that working on all phases of PUG and Pascal News has made me a better person. I had the privilege to experience the processes of organizing, accounting, budgeting, editing, filing, printing, archiving, implementing ideas, pasting-up, publishing, planning, mailing, banking, maintaining mailing lists, juggling details, coordinating events, reading faster and writing better, and talking and working and negotiating and learning with other people!

Thanks

It's the honest truth: we've received hundreds of encouraging and favorable comments about Pascal News. It was truly gratifying to receive nice words this year when our hopes were dim and spirits were down. But then it is only appropriate to thank everyone who contributed material and ideas to Pascal News (by sending them in) and made the whole effort possible. Regular contributors were especially valuable. (As an example there is no reader of Pascal News who doesn't know Arthur Sale. He is a Pascal "folk-hero" because his prolific efforts are accompanied with an unforgettable signature and the end-of-the-earth Tasmanian letterhead.) I've been much less an editor than a collector and organizer of information, and I would like to say "Thanks!" and encourage all of you to keep sending in information no matter how small. Unfortunately, I'm sure we still only know less than half of the news concerning the use of Pascal!

We (especially myself) are indebted to the people whose names are listed below. They volunteered their time, energy, and enthusiasm over the last 4 years directly producing and distributing Pascal News (listed chronologically):

John Strait 1976	Herb Rubenstein 1977
Christi Mickel 1976, 1977, 1978	Arthur Sale 1977, 1978, 1979 (Aus.)
Tim Bonham 1976, 1977, 1978	Jenny Sinclair 1977, 1978, 1979 (Aus.)
Judy Mullins Bishop 1976, 1977 (U.K.)	Rich Cichelli 1978, 1979
Tony Gerber 1976, 1977 (Aus.)	Scott Bertilson 1978, 1979
Carroll Morgan 1976, 1977 (Aus.)	Steve Reisman 1978
Sara Graffunder 1977, 1978	Liz Karl 1978
Jim Miner 1977, 1978, 1979	Jerry Stearns 1978, 1979
John Easton 1977, 1978	Kay Holleman 1978
David Barron 1977, 1978, 1979 (U.K.)	Rick Shaw 1978, 1979
Rick Stevens 1977, 1978, 1979	Tim Hoffmann 1978
Tony Addyman 1977, 1978, 1979 (U.K.)	Rick Marcus 1979

How do we put together an issue of Pascal News?

Invariably the process begins by catching up (1) on the mail. This means opening an accumulation of what used to be 2-4 weeks worth in the early days of PUG to 2-22 weeks worth recently (I'm talking about trays of mail 1 or 2 meters long!). The mail must be separated (2) into new subscriptions, renewals, inquiries for information, changes of address, incorrect payments (returned), purchase orders without prepayment (returned), miscellaneous queries, and material for publication in Pascal News. (To keep our files uniform and organized we manually fill out an All-Purpose Coupon for new subscriptions and renewals; for requests for old backissues, we manually write out an address label.)

The money must be deposited (3) and accounted for (4). New members and renewers must be keyed into the data base (5) and then checked for errors (6). Back issues are mailed (7). The roster increment is run off (8), and the All-Purpose Coupons with tidbit comments are photocopied (9) for the Here and There editor.

The material for Pascal News is gathered together in a pile (10) and then sorted (11) into regular categories (Here & There, Open Forum, etc.). The Implementation Notes section is preprocessed (12) (outlined) and given to the Implementation Notes editors. The Books and Articles section is treated in the same way.

The Articles section is planned and received-dates added (13). The Open Forum section is planned (14). At this point all parts of the issue are attacked (15) at the same time including the subsections of Here & There not mentioned and the Applications section. The cloud which hangs over the process is writing the editorial (16) which delays actual page layout and pasteup of the rest of the issue.

When the camera-ready copy of the editorial and everything else is ready (or nearly so), paste-up with rubber cement on large computer-listing paper begins (18). Each sheet of large paper was previously titled and page-numbered (17) in a typewriter. The completed original is photocopied (19) to produce the 2 copies for PUG(UK) and PUG(AUS) to print from. It is then sent (20) to the printer together with a print order.

Unfortunately, these events don't always occur in this order, thus creating synchronization problems. Needless to say we are always alert for news about Pascal in other journals and from people who call on the phone.

Editor's Contribution

In Closing

As an escape clause, I've always listed: "as well as the ideas behind Pascal" together with "promoting the use of Pascal" as a purpose of Pascal User's Group. We all know that Pascal is not a perfect language, but that it best embodies the ideas of the structured-programming revolution of the 1970's.

Acceptance in the United States has been the icing on the cake and was crucial to Pascal's success as a popular programming language (sorry, ALGOL-68!). So if you stop to think, it is important to note that the Pascal movement in the United States was spearheaded by George Richmond (with some initial help by Lyle Smith, a friend of Niklaus Wirth) at the University of Colorado Computer Center primarily during the years 1972 to 1975.

The effort has been continued by Pascal User's Group via Pascal News by communicating "vast quantities of information" from late 1975 to present. Pascal News and Pascal User's Group (that is, all of us!) succeeded in centralizing authority for Pascal's acceptance, development, and standardization.

What we have done through the medium of Pascal News which was not being done (and probably could not have been done) by any other journal was to openly advocate the superiority of the principles behind Pascal. Perhaps we succeeded in shaking up enough people to accelerate rationality in programming and sensibility in computing by a number of years.

We oversaw a political process and interjected some self-fulfilling propheses to keep the action rolling. Inevitably we all were affected by the spectacular outcome ourselves!

Andy

- 1979/10/21.

DON'T FORGET TO RENEW YOUR PASCAL NEWS SUBSCRIPTION.

digital

October 23, 1979

PUG is not dead!

The first question you all probably asked yourself when you heard the news is: "How will it change?" Well, the answer is: "Not much at all!". It is going to be the same old PUG you grew to love and respect. With the same editorial policy, and the same informal approach to publication. (But it will come out four times a year, that's my only promise!) The only noticeable change will be the Editor and who does all the work. I am not a human dynamo like Andy, so I have had to enlist the aid of many volunteers (see Andy's column) to help me with all the work that used to be done almost single handed. If the next few issues are not as slick as you are used to from PASCAL NEWS, please bear with us. It may take one or two issues to get it right. Number 17 will be out by the end of the year and we hope to have Number 18 published by the end of February.

As a closing note, I would like to ask that all of you start using the new All Purpose Coupon published in this and subsequent issues of the News. It will speed the transition up tremendously between Andy and myself.

See you all next issue. Long live PUG!

Rick

Rick Shaw

RS/cgf

DIGITAL EQUIPMENT CORPORATION, 5775 PEACHTREE-DUNWOODY ROAD, SUITE 100D, ATLANTA, GEORGIA 30342
(404) 252-2600

Introduction to Special Issue

The Pascal Validation Suite - Aims and Methods

by A.H.J. Sale,
Department of Information Science,
University of Tasmania.

1979 July 13

Once upon a whisper-time - so it is said but who would believe it? - long before the Minnipins reached the Land Between the Mountains, the Glocken of Then played upon his bells and the beetle-bores, which had come to infest the countryside, fell upon their backs and waggled their legs for a space and died. But the smell of dead beetle-bores was great, and the bells could do nothing about that, and this I believe.

Pretend-story, told by Glocken to Glocken to Glocken, from Then to Now.
(by Carol Kendall - The Whisper of Glocken)

1. Definition

A Pascal validation suite? What is that? Ignoring the facetious definitions, such as a suite of motel rooms where a Pascal salesman wines and dines clients and promotes the features of the language, there are still a lot of possibilities.

It might be a set of programs that check whether some other (input) text is a valid Pascal program or not. (This may run into the halting problem, well-known to be incomputable). But it isn't. The shortest definition I can supply which describes the validation suite I am talking about is:

> *The validation suite provides programs and procedures whereby the correctness (or otherwise) of a Pascal processor may be tested.*

2. Syntax + Semantics

There are two key words in this definition which have been carefully chosen, and which deserve consideration. Firstly, the definition encompasses a Pascal *processor*, not a compiler. The definition therefore covers processors that compile native machine-code and run it, processors that utilize an intermediate code and an interpreter (for example the P-compiler), direct execution systems and pure interpreters.

But more importantly, the term processor encompasses both the analysis of the source text of a program on an execution system. I am not interested solely in determining that a compiler is "correct", whatever that means, but in determining that the *compiler+machine* pair is correct. To take a very simple example, the following program fails on some processors:

```
b:=true;  c:=false;
if (b=(not c)) then writeln('PASS')
               else writeln('FAIL');
```

What goes wrong? On analysis of the failures, the compiler seems to generate good, correct, code. Giving P-like code as an example, the test compiles to:

```
LOAD  b      {to stack}
LOAD  c      {to stack}
NOT          {logical inversion of c}
EQUAL        {test equality, leave logical result}
BRFL  ...    {branch if false}
```

The flow usually resolves into a failure of the compiler assumptions. The NOT instruction perhaps does not do the transform *true* ↔ *false*, but does a whole-word bit inversion. Coupled with the action of an equality test, of course it may then fail! The bit pattern resulting from inverting the false-pattern is not necessarily the same as the true-pattern.

Resolution of this problem can take several paths, presuming that the machine architecture is fixed. (This assumption is false for interpreters, of course). The simplest resolution is to ensure that every occurrence of the not-operator results in code that allows only the bit-patterns 00....00 and 00....01 to be generated, as is the usual representation of boolean values in more than one bit. In the Burroughs B6700 this can be achieved by emitting the instructions:

```
LNOT         {wordwise logical inversion}
ISOL(0,1)    {isolate the 0-th bit}
```

Another resolution hinges on the manner in which the branch tests are done. Suppose that the machine, when faced with a complex logical expression, generates a sequence of branch instructions which lead eventually to loading a true or false value (or a branch if that is required.)

A third resolution permits other representations for true and false (perhaps including don't care bit positions), but realizes that an unusual representation may require special coding

(a) at conditional branch points
(b) wherever ordering is important (eg. for parameters of ord, succ and pred applied to boolean operands), and
(c) in 'relational expressions' involving booleans, which may be turned into something other than number comparisons (eg. exclusive-or).

For example, in the Burroughs B6700 the representations ??..?0 and ??..?1 would suffice as the conditional branch instructions sense only the right-end bit, and consequently complex code would only be needed for the relatively unusual case where the ordering of the values was relevant. But this is not a treatise on implementing booleans. Suffice it to note that testing a processor involves the semantics as well as the syntax, and the machine as well as the compiler.

3. Ultimate futility or useful weapon

The other carefully chosen word in the definition is *correctness*. Many people have pointed out that testing cannot prove a program to be correct; it can only uncover bugs in it. Since a compiler is a program, and only part of a Pascal processor, is testing therefore an exercise in futility that we should abandon in favour of proving the compiler-machine pair to be a correct implementation of Pascal?

Of course, yes, if we were capable of it. But proof procedures are still human processes, and still subject to error and oversight. Therefore, even proved systems should be subject to testing in order to uncover weaknesses of the proof or oversights. It will be obvious that this is necessary when it is realized that Pascal-P, despite its heritage of careful design, its origin in Zurich, and its extensive testing, still has errors in it which can be detected in most of its descendants.

The validation suite is a set of programs, methodically assembled (and otherwise) which therefore exercise a Pascal processor fairly thoroughly and hopefully uncover many of the flaws in its design or mistakes and deviations from the intended actions.

So much for the theory. How does it work in practice?

4. Conformity

The most obvious set of tests to incorporate are those set down in the Pascal Standard as required of a Pascal processor, or implied to be. These were generated by systematically working through the Standard and wherever it said something was allowed, writing a program to check that it was. All such programs are, of course, standard Pascal.

Since the program *should* execute, it is arranged to print 'PASS' if it works correctly. Some such tests check semantic details, and incorporate run-time checks that lead to failure messages. By the nature of the test, several conformity checks can be included in one test program. Any failure is sufficient to show up a flaw: successful programs exercise all tested features.

However there are three problems: iteration, depth and irregularity.

The first arises with iterative/recursive syntax, or axiomatic recursive semantics. Clearly no program can test all cases of a potentially infinite system! This is tackled by the "one, two, many" principle, named after the primitive enumeration systems of nomad Bushmen in southern Africa. The test will include a minimum case of the construct, probably its successor, and a case which is a small multiple of iterations/recursions, and quite plausible. For example, for identifiers in a *var* declaration:

```
program txxx(output);
var
    onlyone : integer;
    first,second:boolean;
    x0,x1,x2,x3,x4 : char;
begin
    ....
```

Since no processor in a finite computer can provide for infinite recursion/iteration, these tests establish a *prima facie* case that the construct is present and works for small instances up to some unknown limit. To establish that the limit is sufficiently high that it will almost never prove to be a problem - what I have called a *virtual infinity* - another program is written which has a repetition slightly (and only slightly) beyond a plausible maximum. Perhaps 100 cases might be enough for some things, and 20 for other things. Such tests are regarded as not being compulsory, and form part of the *quality* measurement category. (Nevertheless, a processor may fail even a quality test by getting itself knotted, or giving an erroneous diagnostic, or behaving in an unexpected fashion...)

The third problem is irregularity. Successful execution of a test is no guarantee that the same feature will be handled correctly in a different context. Unfortunately there is no way to know all possible context changes that might affect the outcome, and the designers of the test programs have had to draw on their knowledge of machines and implementation techniques to explore this difficult area. A good example is the implementation of type boolean. Clearly, from our earlier discussion, relational operators must be tested separately for booleans and other scalar types.

Another example which maybe should be in the test package, but isn't yet, is the following

```
    csquared:=2.0*alpha*beta,
    if csquared = a*a+b*b then
        {writeln(a,b, sqrt(csquared))};
    ...
```

A particular processor noted that *csquared* was used immediately after its assignment and modified the code to leave this value on the run-time stack to eliminate the fetch. A separate optimizing routine realized that the if-statement had no effect once the debug print was factored out, and deleted the whole of the if-statement code. Result: the stack grew every time this code was executed.... Such interactions require cunning and experience to deduce, let alone devise tests for their presence.

5. Deviance

Besides saying what ought to be allowed, the Pascal Standard says what ought not to be allowed, both explicitly and implicitly. If a processor allows such constructs, it is either a deviant processor and ought to be fixed, or it embodies some deliberate extension which ought to be documented.

Note that it is pointless trying to detect all possible extensions: they form an infinite set! In any case this is not the purpose of deviance tests. Their purpose is to detect the many traps and failures to enforce reasonable restrictions which compiler implementors can unwittingly place in the way of users.

Some deviance tests are directly suggested by the Pascal Standard. Good examples are afforded by the restrictions placed on for-statement control-variables. But by far the larger group are not. These rely on the test program writers' intelligence and knowledge of compilers and on the ability or experience to recognise possible problem areas. Two examples will illustrate this.

The first was derived from some experience with an optimizing compiler which generated different code for

```
j div k
```

depending on whether *k* was a constant power of two or not, or so the documentation said. This immediately challenged the hardware designer in me to check this out, and challenged my software designing side to prove it correct. Since the optimization relied on a property of the integer representation, it was potentially possible for it to fail. Sure enough, on this compiler, it did: *j div k* returned different results for the same value of *k*, solely depending on whether it was a constant or not.

The second case grew out of a knowledge of the workings of the compatibility algorithm of most Pascal compilers, and some strict wording in the Standard. Strictly

```
type
    digitstring=packed array[1..10] of '0'..'9';
```

is not a 'string' type, and not compatible with the usual quoted string constant. Thus

```
var
    d : digitstring;
begin
    d:='0123456789';
```

is not correct Pascal: it is a deviation or an extension. Of course, compilers that allow it as an extension ought to do it properly (ie. consistently with the axioms of Pascal), so this naturally gives rise to the attempt:

```
d:='ABCDEFGHIJ';
```

which is plain impossible.

Enough of examples. It will be obvious that every deviance test program is non-standard, and that they should mostly fail to execute to completion. Unlike conformance tests this means that each deviance test tries one and only one deviation. Otherwise one failure might mask another, more serious. Conformance and deviance tests make up the majority of the tests (71% of the current suite), and do the bulk of the exercising of the compiler.

6. Errors and implementation features

The Pascal Standard also specifies a number of situations which are specified to be errors, implementation-dependent, or implementation-defined. Each of these gives rise to a test or tests that evoke the feature so specified.

In the case of errors and implementation-dependency, the purpose is to enable documentation of the error-handling capability of the processor. Hopefully all errors are detected; in practice some of them survive into production software and may even become part of programmer's assumption kit... All such programs are directly suggested by the Standard, evoke one and only one error, and are standard Pascal except for the error.

Naturally, implementation-defined features should be documented, and the corresponding tests attempt to do this by a variety of techniques. Some, like printing the value of maxint, are simple standard Pascal. Others, like trying to detect the significance limit of identifiers, rely on assumptions about the nature of the lexical analysis and scope and are clearly not standard. The inclusion of these tests is really to improve the documentation of Pascal processors.

The number of tests in those categories is small and unlikely to change much. The specific things they test are more or less fixed by the Standard.

7. Quality

All the preceding tests address the problem of whether the processor conforms to the Pascal Standard and, where necessary, how. It is also relevant to ask whether the implementation measures up to some quality standards, and the validation suite therefore contains tests which attempt to assess *quality*, however you define it.

As you might expect, these tests form a motley group. Some attempt to assess whether the processor provides some hard limits that are likely to annoy in a particular direction, or whether the limits are at virtual infinity. Others attempt to assess accuracy of mathematical functions such as sqrt(c). What they all have in common is a requirement for greater analysis of the results and a value judgement.

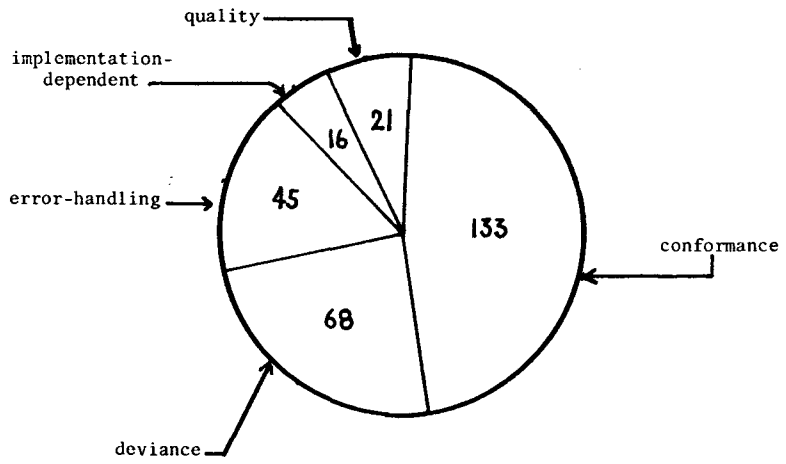
Of course, a processor may fail even a quality test by totally mishandling it. For example, if the compiler goes into an infinite loop, or gets its lexical levels screwed up. But this is not the intention, and is rare.

This group of tests is likely to have the greatest growth rate in future revisions of the validation suite. For obvious reasons, the first release version has concentrated on correctness, and while there may be some growth in tests for conformity and deviance as we understand the problem better, it is already apparent that the suite lacks quality tests in several areas:

(a) tests that can be timed,
(b) tests whose space utilization is measurable,
(c) tests of compiler diagnostic ability,

and there are no doubt more.

The breakdown of programs in version 2.1 by class is shown below.



BREAKDOWN OF PROGRAMS BY CLASS

8. Feedback

No validation suite can ever be perfect. Since its task is infinite, one can only approach perfection with varying degrees of success. Consequently, it is very important that a continuous revision of the suite is maintained, with three main objectives in mind:

(a) Removal or modification of tests that do not agree with requirements of the Pascal Standard.
(b) Addition of new tests arising from experience and context changes, and therefore unexpected interpretations.
(c) Expansion of quality control tests aimed at improving the quality of Pascal processors in all dimensions of choice.

Suggestions for categories (b) and (c) are therefore welcome. Complaints about category (a) are even more important. All will be read, digested, and carefully considered, though clearly some purely idiosyncratic tests may not make it into the package.

There is no formal bug-reporting service. Simply write to me at the following addresses:

up to January 1980

Professor A.H.J. Sale,
c/- Professor D.W. Barron,
Department of Computer Studies,
The University,
Southampton, England SO9 5NH
UNITED KINGDOM.

from January 1980

Professor A.H.J. Sale,
Department of Information Science,
University of Tasmania,
GPO Box 252C,
Hobart, Tasmania 7001
AUSTRALIA.

Please enclose a listing and any other information relative to the report/request.

But this is not enough! Simply casting the validation suite out into the wide world calls out for more. My own curiosity is reason enough, but in addition it has become apparent that overseeing the results of other people's passes of the validation suite against a processor can be extremely valuable.

Accordingly I ask you to write to me if you are prepared to send documentation on a validation run for a processor I haven't already collected. I will let you know if we already have data on that system. The documentation I would want is:

(a) a complete set of listings of runs of the validation programs, annotated to explain any obscure effects.
(b) a validation report, similar to the one produced for our own compiler.
(c) an accurate identification of the processor (compiler source and date of acquisition or version, machine identification). A manual would be useful too.

With help we can begin to assemble a comparative list of processor performance, and watch the way the situation evolves.

9. Utilization

I hope to see copies of validation reports for processors of significant interest in future editions of Pascal News. Obviously implementors will want to fix minor bugs, but may baulk at fixing difficult ones or publishing the results. Some comparative results published by responsible users will assist readers of the News to assess comparative merits of compilers, may bring some collective user pressure to bear on the implementors/maintainers to fix even the more persistent and difficult bugs, and will assist the Users Group to assess which systems are still active or most in use. I hope to see the validation suite being used by at least two groups of people: the implementors/maintainers and the users.

Implementors of course will use the package to check out their product; maintainers would be well-advised to do the same after each major revision. They will also be able to compare their efforts against other compilers more easily, and a bit more competition will be good for both suppliers and users.

Users will be able to use the package to bring pressure on suppliers to conform to the draft standard, and therefore use the package to evaluate the relative merit of two or more systems. Also, because the validation suite is virtually the Pascal Standard cast into test programs (with some reservations), access to the essential concepts of the Pascal Standard will be made more clear by inspection of the conformance and deviance tests.

Naturally, as the draft Pascal Standard stabilizes, the agreed resolutions will be embodied in further tests, thereby both providing an enforcement technique, and a method of publicizing the agreement. For example, this may happen in the area of the *pack* and *unpack* procedures which are perhaps overly restrictively defined.

10. A Confession

Once upon a time, long ago, programmers who could write tricky code were prized. This pleased them, because they were being well-rewarded for doing what they enjoyed: creating private masterpieces of complexity on a small scale. Any modern abstract artist would understand the feeling.

Now, programming has completely changed. We strive to write correct programs together with their proofs. The creativity, and the accompanying surge of pleasurable sensations on completion of a particularly difficult task, are still with us, but in a different form, and with different nuances and vibrations. I enjoy this tremendously when administrative chores allow me to indulge myself in writing programs, and I wouldn't give up the advances we've made in programming methodology for anything.

And yet, there is still that fascination with those tricky, nasty jewels we created. I must therefore confess that I believe that when the last tricky programmer on earth expires his orher last breath over a soggy sheet of paper, he or she will probably be writing a validation program. It seems to be the last refuge left for contorted thought, for to paraphrase a famous fictional detective (I think it was Hercule Poirot), how can you discover what crime has been done unless you can put yourself into the frame of mind of the criminal?

11. Some favourite tests

I must confess to having some favourite tests, which either are utterly useless and indulge a peculiar sense of humour, or are particularly devastating. I share them with you in case you might, too, share this sense of humour.

(a) Syntactic

Test 6.4.2.3-1
The interesting thing here is the peculiar scalar

```
type
    singularitytype = (me);
```

which doesn't seem to be useful for anything, though you can assign to it, test it, etc. The apparently similar (test 6.4.1-1):

```
type
    purelink = ↑ purelink;
```

is less amusing because it does have at least two distinct potential uses.

(b) Context-sensitive

Test 6.4.3.3-4
Here the use of a field-name which is already defined trips up a number of compilers. They don't wait to find the colon before rushing into analysis. Most Pascal-P compilers inherit this one.

(c) Scope

Test 6.2.2-2
The sheer perversity of being able to write

```
if true = false then writeln ('PASS')
```

is delightful.

(d) Execution

Test 6.8.3.9-7
The beauty of this one lies in two aspects: many processors completely fail it, and our compiler turned out to pass it quite unexpectedly. It comes as a surprise to devise a test that you confidently expect to fail on your own implementation, and then the thing makes a fool of you by working.... And then you have to work out how it outsmarted you. To add to its perverse charm, it usually turns out to have a simple resolution which, though unattractive, requires only that *maxint* be reduced by one.

(e) Errorhandling

Test 6.6.5.2-6
I must admit that the attraction of this one lies both in the weird variety of effects it can evoke, and to some extent by the clarity with which the aliasing problem points an accusing finger at the file buffer concept. Interestingly, a simple restriction would remove the problem, by simply not permitting file-buffers in such contexts, but it would introduce more irregularity....

Acknowledgements

I wish to acknowledge the great debt the Validation Suite owes to a multitude of people. Those I single out for special mention here contributed especially, but there are many other contributors for which there is not sufficient space.

Brian Wichmann : for initiating the project, for carrying it out throughout the first phase, and for many insights and tests, and as joint author.

Andy Mickel : for encouragement to continue, despite over-runs.

R.D. Tennent : for many critical comments on semantics.

Roy Freak : for patient and hard work in assembling around 300 programs to a consistent style and putting up with my nit-picking and niggling.

Nigel Saville : for diligence in interpreting often partial instructions and creating a large number of provably correct (or provably incorrect) programs in the rewriting phase.

Jenny Mizielinski : for laboriously and carefully typing (seemingly endlessly) abtruse documents full of mysterious numbers, each of which was highly significant.

The Sale Family : for putting up with grunts and groans, and with listings strewn around the sitting room.

Of course, to all my correspondents who contributed unknowingly must go a special kind of thanks. Without your stimulation, the Validation Suite might not have had the firm basis it now has.

Arthur Sale
Tasmania

1979 August

THE PASCAL VALIDATION SUITE
Version 2.2

DISTRIBUTION INFORMATION AS AT 20 AUGUST 1979

Revision History

Version 2.0 was the first release of a completely rewritten package which was based on earlier work by B.A. Wichmann and A.H.J. Sale. This earlier work is considered to be version 1, and is now obsolete.

Version 2.0 was indexed to Working Draft 3 of the Pascal Standard as published in Pascal News #14. Version 2.1 fixed up a few bugs detected after release. Version 2.2 is altered in indexing to refer to the draft ISO Standard document ISO/TC97/SC5/N462, and incorporates more tests which facilitate the timing and measurement of quality in Pascal processors. Subsequent revisions will be issued when either detected errors in the package require a revision, or when a new version of the draft standard is released.

Purpose

The validation suite is provided to exercise Pascal processors and to determine by testing whether the processor conforms to the requirements of the Pascal Standard or not, and to provide a common set of programs for documenting implementation-dependencies and quality of implementation. It is strongly oriented around the draft Pascal Standard, and tests are suggested by that document. A few proposed tests have been omitted because it has been suggested that the draft Standard will be revised in that area, but these are few. It follows therefore that any revisions of the draft Standard will cause revisions of the validation suite. The suite currently contains over 300 programs.

Acquisition

To acquire a machine-readable copy of the validation suite, apply to one of the distribution centres. It will be necessary to fill in a software licence, and a small fee (around US$50) is charged to cover costs. The fee covers the supply of a magnetic tape, the copying of the validation suite onto the tape, airmail postage to the address provided, and a limited notification service relating to later releases or inaccuracies detected in the suite.

Restrictions

The conditions of release prohibit the distribution of the package to third parties so as to limit the growth of unauthorized and inaccurate versions. The likely incidence of change if the draft standard is revised will show the desirability of this requirement. However, no restriction is placed on the use of the package for validating Pascal processors, for benchmarking, for acceptance tests, for preparing comparative reports, and similar activities, nor on the distribution of the results of such use.

The validation suite is expected to be widely used and distributed, not restricted to a small subset of the user community.

Feedback

No special reporting mechanisms have been set up. However, the authors will attempt to produce revisions of the validation suite and distribute them to the distribution centres as necessary, and to publicize their availability.

Information relating to a pass of the validation programs against a particular processor would be welcome at the University of Tasmania, but it must be understood that the authors cannot provide full reports on all listings provided.

Of particular value would be any tests that produced entirely unexpected results which may be of wider interest, or which are without any reasonable explanation from the user's point of view. In some cases the authors may be able to deduce the likely cause, or recognize an epidemic of common flaws. Any correspondance which points out an error in the classification of the programs in the suite, or in its construction, or suggests a new test, would be most welcome.

Address for suggestions or complaints:

September '79 - January '80:

Professor A.H.J. Sale,
c/- Department of Computer Studies,
The University,
Southampton, England SO9 5NH
UNITED KINGDOM.

February '80 onwards:

Professor A.H.J. Sale,
Department of Information Science,
University of Tasmania,
GPO Box 252C,
Hobart, Tasmania 7001
AUSTRALIA.

Distribution Format and Addresses

```
Character set

 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~

THIS TAPE CONTAINS 4 FILES :

FILE 1:
     THE CHARACTER SET USED IN THE VALIDATION SUITE AND DETAILS
     OF THE TAPE STRUCTURE.

FILE 2:
     THE SKELETON PROGRAM WHICH IDENTIFIES EACH TEST PROGRAM

FILE 3:
     THE DOCUMENTATION OF THE PASCAL PROCESSOR VALIDATION SUITE

FILE 4:
     THE SUITE OF PASCAL TEST PROGRAMS

EACH FILE CONSISTS OF 80 CHARACTER RECORDS.
A SEQUENCE NUMBER IS IN COLUMNS 73-80 INCLUSIVE.
FILEMARKS SEPARATE EACH FILE.
TWO FILEMARKS AT END OF TAPE.
THE TAPE IS UNLABELLED.

TAPES SUPPLIED BY THE UNIVERSITY OF TASMANIA AND DR B.A. WICHMANN
HAVE THE FOLLOWING FORMAT:

  15 RECORDS PER BLOCK.
  EACH RECORD IS WRITTEN IN ASCII.
  THE TAPE WILL NORMALLY BE WRITTEN AT 1600BPI, PE TAPE.

TAPES SUPPLIED BY R.J. CICHELLI ARE WRITTEN AT
  800 BPI, NRZ, ODD PARITY, 9-TRACK
  AND 1) EITHER ASCII OR EBCDIC
      2) 1,10,20 RECORDS PER BLOCK

IN EITHER CASE, OTHER TAPE SPECIFICATIONS MAY BE ARRANGED BY CONTACTING
THE DISTRIBUTERS.

FILES 2, 3 AND 4 CONTAIN BOTH UPPER AND LOWER CASE LETTERS.
FILE 1 CONTAINS UPPER CASE LETTERS ONLY EXCEPT FOR THE SPECIFICATION
OF THE CHARACTER SET.

VERSION 2.2

JULY 1979
```

```
ADDRESSES:
  DISTRIBUTION IN AUSTRALIA, NEW ZEALAND AND JAPAN.
  ALSO ANY GENERAL CORRESPONDENCE, CORRECTIONS, QUERIES.

    PASCAL SUPPORT,
    DEPARTMENT OF INFORMATION SCIENCE,
    UNIVERSITY OF TASMANIA,
    BOX 252C, G.P.O.,
    HOBART 7000
    TASMANIA
    AUSTRALIA.
    61-02-230561 EXT 435

    DISTRIBUTION COST: $A50

  FOR DISTRIBUTION IN NORTH AMERICA -

    RICHARD J. CICHELLI,
    C/- ANPA RESEARCH INSTITUTE
    BOX 598
    EASTON, PA. 18042
    U.S.A.
    (215) 253-6155

    DISTRIBUTION COST: $US50

  FOR DISTRIBUTION IN EUROPE -

    DR. B. WICHMANN,
    NATIONAL PHYSICS LABORATORY,
    TEDDINGTON, TW11 0LW
    MIDDLESEX,
    ENGLAND
    01-977 3222 EXT 3976

    DISTRIBUTION COST: NOT KNOWN
```

DEPARTMENT OF INFORMATION SCIENCE
UNIVERSITY OF TASMANIA
BOX 252C, G.P.O.,
HOBART 7001.
TASMANIA
AUSTRALIA
